AUSTRALIAN OS9 NEWSLETTER

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-1284
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
SUPPORT	Brisbane OS9 Users Group	

Addresses for Correspondence

Editorial Material:

Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

Volume 5 August 1991 Number 7

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 5 Number 7

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

Welcome to our August edition of the usergroup newsletter which is the last for the current subscription year. This also brings us to the end of the third year of the National OS9 Usergroup since being revived in sunny Queensland. Please use the application form attached to renew your subscription for the coming year as this will ensure that we have current information about you and your OS9 system.

I must say that we are just a little suprised at the number of enquiries received recently from people just entering the OS9 world on a CoCo3 platform despite the obvious difficulties in obtaining the Microware 6809 OS9 level 2 for the Color Computer. By the way, if you know of anybody wanting to purchase the OS9 L2 operating system for the CoCo3, it is available from:-

COCOPRO!
1334 BYRON
YPSILANTI MI 48198
U.S.A.

Price $US 34.95 Plus Shipping & Handling

Many of us, however, are looking for the "where to next". OSK will probably be the logical upgrade for many of us CoCo users with the big question still being:- What new computer should I invest in?

This newsletter contains an update on the status of the MMI development "as is" released by IMS (Interactive Media Systems) early July. The MMI certainly promises to be an exciting OS-9 platform. We have devoted some three pages to this "release" in order to bring you the latest information and trust that it is of interest even if you have not been tempted to add an MMI to your wish list.

It is also interesting to note that Microware's generic term OS-9 refers to what we have known as OSK or OS-9 68000 and in fact has done for some time. The magazine "Pipelines" produced by Microware no longer has any reference to OS-9 6809 which they dropped some time ago, to be up to date, the term OS-9 really refers to the Microware operating system for the 68000 family of processors.

OS-9000 What about OS-9000 for the 386/PC and clones? Microware's recently released operating system for 386 based machines has very similar architecture and features to OS-9

A direct quote from Microware's handout 'Questions & Answers for serious 386/PC users' "OS-9000 is a modular real-time multi-user, multi-tasking operating system designed to unleash the power of your 386/PC compatible. Now you can have powerful VME-style software for your PC...at a price you can afford. Featuring a ROMable real-time Kernel, file managers, utilities and networking options. OS-9000 offers you the ultimate real-time software development platform for your 386 hardware".

For graphics support you will also need RAVE (Real-time Audio/Visual Environment) which, among other things, offers video animation and CD-quality sound.

For more information, contact the Australian Microware agent:-

Microprocessor Consultants P/L
P.O. Box 312
Avalon Beach NSW 2107
Phone 02 974 4917

Will we ever become the National OS-9000 Usergroup? I don't think so, but here is an alternative which may be attractive to some. I have no details of prices, but expect that one would need more than just a hobby in mind to enquire.

We trust that you find something of interest in this newsletter and remind you that contributions for inclusion are most welcome. YOU can change the content simply by sending your article to the editor. This Usergroup is operated on a non-profit basis with the aim of assisting OS-9ers, so if you feel that you are not able to contribute, please let us know what you would like to see included in future newsletters.

Cheers, Gordon.

Where we stand on the MM/1
by Paul Ward

To help keep you up to date, here is a file that tells you: what we are trying to accomplish, what we are accomplishing, and what we have done to improve the MM/1 and IMS.

IMS, Inc. was created to bring you the MM/1 and affordable OSK software. The MM/1 was and is a group project thought up by this community. Many of you contributed ideas, and you still have a vital role to play by actively participating in providing constructive feedback to Interactive Media Systems, Inc. All input is considered and appreciated.

Before we get started, let's evaluate where the MM/1 is now.

First, the two board kit is still being sold. The price has been raised to $975, so the $875 special price is now over. Also, our case price has been raised from $100 to $125. With the kit comes with some additional free software, enhanced windowing software, new documentation and with some technical literature.

Second, more technical literature is being prepared that will also provide specific ideas on projects with the MM/1.

Third, improved SCSI drivers are in beta test that bring the MM/1's SCSI subsystem to workstation level speeds -- 1.8 Megabytes per second transfer rates on a Quantum Pro drive.

Fourth, updated hardware and software is beginning to be distributed to developers.

Fifth, the MM/1 kits are shipping. We had some delays after the Fest with a mask error Western Digital introduced into their SCSI host adapter chip. Kits WITHOUT the I/O board are shipping now, and have been shipping for weeks. We should have well over one hundred MM/1s in the field by the end of July, and shipping will continue through August until we are caught up.

The mask change of the SCSI host adapter chip was introduced only in later models by Western Digital. Three hundred chips from an older mask have been reserved for IMS. These are being tested this week. An update will be available soon on this information service, or at 202-232-4246 between the hours of 2 pm and 5 pm, Monday through Friday. (We will be closed July 4 in honor of Independence Day.)

What IMS is trying to accomplish.
==========

Goal: Solve the Software/Hardware Paradox

You can't have software that is good quality unless the software vendors have a large market to sell to. Yet you won't buy a computer unless it has good software.

This dilemma has been stated in this and other forums as the biggest challenge facing any company offering an upgrade to the Color Computer.

IMS is the only company to have taken this challenge head on. We started tackling this problem as long as two years ago. No other company has been as active for as long in creating this new OSK market.

Our first step was to map out methods for creating quality software quickly. Two major portions of that step are done. The MM/1 supports a windowing system strikingly compatible with Color Computer OS-9 windows. And the MM/1 supports a C graphics library that allows the easy porting of MultiVue based software to the MM/1. Other steps are in place now, and new software will start pouring in for the MM/1 by Christmas -- without a lot of effort from developers!

With new software available for the MM/1, now users have confidence to purchase it. The software developed on the MM/1 sometimes will work on the generic OS-9/68000 platforms offered by others including Peripheral Technologies, Tadpole, GESPAC, Mizar, GIMIX, and others. Sometimes MM/1 software will NOT work on these platforms. Software will NOT work on other OSK platforms when a developer take advantage of the MM/1's Sun(tm) workstation-style color architecture, or when a developer's application depends on the variable sampling rate of our built-in sound.

It is up to the developers to make their product work on other OSK computer systems. These developers will make changes to their product

when they believe sales on the other platforms will provide a payoff for their work.

Goal: Solve the distribution challenge

IMS has a high goal: to sell not dozens, hundreds, or thousands of computers, but to sell tens of thousands.

The reason lies in a need for long-term success. You as a customer need good software and hardware for you computer for as long as you choose to use your computer. For a programmer to support himself, he needs to sell you good software.

So, you need developers, and he needs you -- but you only need one of him. The programmer needs enough MM/1 customers to make a living.

Let's do some quick arithmetic. If $30,000 per year is a reasonable salary, then the programmer needs to sell 1,000 units of a product at a net profit of $30 each, per year. If one in ten of MM/1 owners buys a particular product (10% penetration is actually the sign of a successful product in a small market), then the number of MM/1 owners must be 10,000.

Now, you can figure out how many computers need to be shipped per week to meet that goal -- that's 200 units PER WEEK. That's why IMS has a full-time staff now and will expand it in the future. And what about IMS's sales team? How large does that have to be to field the 100,000 phone calls and letters required to generate 10,000 sales?

This is a serious bottleneck, and IMS is apparently the ONLY company with plans in place to solve it.

Goal: Solve the mainstream challenge

For the MM/1 to have wide success (and thus create a wide market that benefits each of you), it needs to survive the hardware and software reviews of major magazines.

Every effort is being made to make that possible. The first step is the elusive FCC approval. The next step is compiling technical documentation and user documentation that sets it goals at educating and training a new OS-9/68000 user. The usual Microware manuals will not suffice.

Also, promotion and advertising out of the Color Computer market is essential. IMS has already taken steps in this direction.

Plus, connectivity and interoperability is a challenge ahead of IMS and the MM/1. By connecting to PCs, Macs, and mainframes, the MM/1 becomes another tool for business and industry.

What IMS has accomplished.
==========

We have solved the Software/Hardware Paradox.

Tools are here now to help create Color Computer OS-9 compatible software. Tools will be available in the Fall to ease the porting of UNIX software. Tools will be available by Winter to ease the porting of BASIC, QuickBasic, Mbasic, and BasicA programs to the MM/1.

Plus, software developers have already created new products for the just-shipping MM/1. By Fall, sixty developers will be actively creating new software for the computer.

No computer has ever generated this level of anticipation, excitement, and NEW PRODUCTS. Thanks go to our supporters, who made it possible.

Already available are a paint program, LHarc dearchiving program for Amiga files, Stuffit dearchiving program for Macintosh files, font editor, new text editor with printing capabilities, send fax, notepad/calculator/scheduler, disk utilities, sound playback applications, Autodesk Animator flicker animation player to play back PC-based animations, C grachics libraries to ease the porting of Multivue programs.

On the way is a point-and-click database, a checkbook program, sound sample editor, new drivers for ultrafast hard drive access, file copying and deleting utilities using a mouse, SCSI digitizer for grabbing frames from a video camera or VCR, teleconferencing, electronic mail, multimedia database, multi-line BBS, and more.

No other OSK system has been the host to this much development activity, ever. Helping us out

are people like Kevin Darling, Kevin Pease, Mike Haaland, Bert Schneider, Scott Griepentrog, Mike Guzzi, Dave Bever, Mark Griffith, and others, including the staff at CompuServe, Microware Hunter Systems, Tundra Software, and Microsoft.

On the way to help port software is the Gnu C compiler and C++ object-oriented compiler, to be tested by IMS and its developers, then distributed with sample software. This compiler will greatly ease the porting of UNIX software to your MM/1.

We have solved the Distribution challenge.

Without going into details that would reveal how we have this advantage, IMS has already begun to put into place an infrastructure that will generate the number of sales required to keep your MM/1 well-supported.

First, we have installed representatives for the MM/1 across the United States. By the end of July, a dozen will be in place in key geographic regions. By December, IMS hopes to double this number. Their locations are strategically located to provide access to our carefully selected markets.

By the middle of August, IMS will announce the names, locations, phone numbers, addresses, and other information about these representatives.

Second, IMS has begun setting up retail operations in some strategic areas. This process is slow on purpose -- we anticipate a manageable number of retail outlets in place by October 1.

Third, IMS has recently greatly improved its systems for communicating with representatives, developers, and end users. This new systems will be incrementally brought into effect over the next six week.

The result of these accomplishments is that a high number of leads can be generated without a lot of sales overhead at IMS, keeping our responsiveness high and overhead low. That saves you time and money. With a high number of qualified sales leads, IMS can help the representatives and retailers educate their potential customers through literature, videos, demonstrations, and visits by IMS staff. These leads will yield sales, and sales allow IMS to continue to bring new software and hardware to the MM/1.

We are solving the mainstream challenge.

IMS has passed another step of the FCC approval process. No nationally reknowned magazine will review a machine that is not FCC approved. Nor is it legal to sell computers that are completely configured for use in the home without FCC approval (this is why IMS sells kits).

In addition to the FCC approval IMS is continuing to seek, another unique thing we are doing is providing information to national magazines and newspapers. Over the next three months, expect to see some national coverage on the MM/1.

And once we get through our shipping backlog, IMS will be appearing in advertisiments to continue our growth.

And for your information ...
==========

Just in case you thought we were sitting on our hands, here is a list of what I, Paul Ward, have done in the last month:

* Coordinated the showing of the MM/1 at two new clubs
* Prepared a new videotape showing MM/1 Extended features
* Sent proposals and correspondence to three universities, who seek the MM/1 as a courseware tool
* Send software and hardware to new developers
* Sent software and hardware to old developers
* Worked closely with StG and The OSK'er to ensure that a complete MM/1 system is reviewed (unlike the system that was reviewed in the latest issue)
* Compiled technical information that clears up StG's misunderstandings
* Prepared literature for retailers (still in progress)
* Prepared literature for representatives (almost done)
* Created new software press releases
* Designed our software catalog and began a hardware catalog
* Tested new device drivers
* Had meetings with three investors
* Began version 3.0 of the MM/1 User Guide
* Compiled suggestions from current MM/1 owners
* Sketched out and started filling out the IMS Road Map for IMS developers

* Set up three new IMS representatives and two new IMS developers
* Tested a new mouse
* Implemented new customer support services
* Began scheduling mailings through the Fall
* Met with and/or talked to representatives from Smith College, Amherst College, University of Massachusetts at Amherst, North Carolina State University, University of Arkansas, University of Texas, and others
* Talked with robotics and data acquisition companies in Texas, Massachusetts, Virginia, and North Carolina
* Arranged for new software for the MM/1 to be tested
* Arranged for a new publisher for "Start OS-9"
* Sent a mailing to CD-I and CD-TV development firms
* Organized some developers to ensure they have new products this summer
* Began architecting networkable multimedia

I am sure I left off quite a bit. Now, multiply last month's activities by three (the number of full time staff at IMS), and you've got an active company working very hard.

If any of you have questions or feedback, once again I'd like to say we are delighted to hear from you.

Best regards,

Paul K. Ward

Interactive Media Systems, Inc.
(+1) 202 232 4266
between 2 pm and 5 pm, Monday through Friday

oooooooooo0000000000oooooooooo

A Basic09 Tutorial
by Bob Devries

Ha! I'll bet you thought you got rid of me with the last article I did on Basic09 conversions! Nope! Here I am to (de)bug you some more. This month I will start looking at the GFX2 commands. First I will cover those that appear in the Basic09 Manual, and give some example programme code to show how it is used. Then I will tell you about the extensions that were added to GFX2 by Kevin Darling. The section of the manual to read starts on page 9-31 of the Basic09 manual.

I will start off by describe the antics needed to create a graphics window so that you can actually use some of the GFX2 commands. First, decide which type of graphics window you want. I will use a type 8 graphics window. Here is how you open the window.

```
OPEN #wpath,"/w":UPDATE
RUN GFX2(wpath,"DWSET",8,0,0,40,24,0,1,2)
RUN GFX2(wpath,"SELECT")
```

Here's what it all means! First, I opened a path (using an INTEGER variable wpath) to the GENERIC window descriptor "/w". I did this so that my programme will work no matter what window descriptor is in use, as long as one of them is available. If I had written the programme like the example on page 9-87 of the manual, and "/w2" was in use, my programme would have quit with an error. The MODE is UPDATE (not WRITE as in the example), so that I can get inout from the window as well being able to draw on it.

Next I used the DWSET call to tell OS9 what size etc to make the new window. The numbers following the DWSET word are:

8 is the window type. This gives a 320 x 192 16 colour graphics screen.
0,0 is the location of the top left corner of the window.
40,24is the width and height of the window in characters.
0,1,2are the foreground, background and border palette register numbers.

OK, so now I have created the window, but one thing is still missing....I can't see it yet. The next command, SELECT, takes care of that.

Right, now we can start to use some of the graphics commands. The ARC command draws an arc (part of a circle or ellipse) on the screen. Here's a sample:

```
run gfx2(wpath,"ARC",50,20,100,10,1,0,1,0)
```

I must admit that it took me quite a while to work exactly how the numbers in this command worked, but I think I've got it. Here goes. The first two numbers are the location of the centre

point of the ellipse. The next two numbers are the X and Y radii of the ellipse. So far so good. Now the next four numbers form the two sets of coordinates which, when you draw an imaginary line through them, form the start and end points of the arc where the ellipse touches that line. This is approximately only, probably due to inaccuracies in 8 bit maths. The manual says: 'ARC begins drawing from the point on the screen closest to the first set of coordinates (xcor1, ycor1). It stops at the portion of the screen closest to the second set of coordinates (xcor2, ycor2).'

If you try something like this:

```
run gfx2(wpath,"ARC",20,10,10,20,30,50,27,15)
run gfx2(wpath,"LINE",30,50,27,15)
```

You will find that the line is parallel to a line drawn through the two ends of the arc. Have a play with that, and try the example on page 9-51 of the manual. It works as long as you add the code to open the window around it. By the way, to close the window and return to your normal screen, this is the code you should use:

```
run gfx2(0,"SELECT")
run gfx2(wpath,"OWEND")
close #wpath
```

The next GFX2 command in the manual is the 'BAR' draw. This one will draw a filled box like the one in RSDOS that goes 'LINE(x,y)-(xi,yi),PSET,BF. The parameters passed are exactly the same. This is what it looks like:

```
run gfx2(wpath,"BAR",50,20,100,130)
```

So, let's have a look at some more graphics commands. I guess the box draw should be next. It goes like this:

```
run gfx2(wpath,"BOX",xcor1,ycor1,xcor2,ycor2)
```

If the first set of X and Y coordinates is left out, the box will be drawn from the current graphics cursor position, where you left off drawing with the last command.

The circle command, unlike RSBASIC, ONLY draws a CIRCLE! So it is simpler than the basic one:

```
run gfx2(wpath,"CIRCLE",xcor,ycor,radius)
```

Again if the X and Y coordinates are omitted, the circle is drawn at the current graphics cursor position.

The draw command is somewhat different from the basic one, in that its direction sub-commands are compass points, instead of arbitrary letters. So, N5 is north(up) 5 pixels. S8 is south(down) 8 pixels, and SE10 is south-east (down-right) 10 pixels. As well as that, the orientation of the compass may be rotated by 90 degrees at a time, so A0 is normal, A1 is 90 degrees, A2 is 180, and A3 is 270 (clockwise, of course!). Next we have the draw relative command. U23,45 will draw a line from the current position to coordinates 23,45 without changeing the cursor position. With this, it would be fairly easy to implement the 'rays' draw that is so popular with drawing programmes. Simply set the cursor to the required centre point, and then draw lines, using the 'draw' command, out to the mouse location. Here's a bit of sample code.

```
repeat
run getmouse(wpath,mouse)
run gfx2(wpath,"PUTGC",mouse.AcX,mouse.AcY)
until mouse.cbsa <> 0

xcor=mouse.AcX
ycor=mouse.AcY

repeat
run getmouse(wpath,mouse)
run gfx2(wpath,"PUTGC",mouse.AcX,mouse.AcY)
U$="U"+str$(xcor)+","+str$(ycor)
run gfx2(wpath,"DRAW",U$)
until mouse.cbsa=0
```

I'll try to explain this programme sample. Please note that not the whole programme is depicted here, for example, the procedure 'getmouse' is not shown, also the variable mouse is a complex variable set up with the TYPE command. The first repeat-until loop is used to mark time, and move the graphics pointer around (PUTGC) until a button is pressed (the left one). Next a new repeat-until loop draws lines from the point at which the button was pressed out to the current location of the graphics cursor, until the button is released. Unless you know how the rest of the code works, (and you don't, else you wouldn't be reading this) don't try to use this code. I will supply a full working example in a later article.

These next few commands are used for text printing, except for the bell command, which, of

course, can be used at any time.

```
run gfx2(wpath,"BELL")
run gfx2(wpath,"BLNKON")
run gfx2(wpath,"BLNKOFF")
```

The first sends a beep to the speaker, the second will set any following text to blinking. NOTE! this only works on a TEXT window. That is, only types 1 and 2 screens. Blinking characters are NOT supported on a graphics screen. The last example above is the blink off call.

This next one is only for GRAPHICS screens. It changes the character set to BOLD printing.

```
run gfx2(wpath,"BOLDSW","ON")
```

and off again...

```
run gfx2(wpath,"BOLDSW","OFF")
```

I will now take some commands out of sequence, but since they belong together, I think it is justified. The commands are used to change the foreground, background and border colours respectively. Here they are.

```
run gfx2(wpath,"BORDER",palreg)
run gfx2(wpath,"COLOR",fore,back,bord)
run gfx2(wpath,"PALETTE",palreg,colour)
```

In the first one, the variable 'colour' is a palette register number, that is, a number from 0 to 15 inclusive. Be careful, the manual seems to suggest (on page 9-58) that it is the colour in the border palette register that is changed, but this is not so. The second line also uses variables to represent the palette registers to use for the foreground, background, and border register. Here is the call you may use to change the actual colour in the individual palette registers. The first value is the palette register number (0 to 15), and the second is the colour value to place in it (0 to 63).

Well, that will be enough for now, next time I may present a complete working set of examples for you to type in and run. Until then, keep practicing writing Basic09 programmes. It is really not very difficult. By the way, if any of you have any problems, or don't quite understand how to do a particular aspect of Basic09 programming, please write to me care of the newsletter, and I will answer them if I can.

Regards,
Bob Devries

oooooooooo0000000000oooooooooo

A C Tutorial
Chapter 3 - Program Control

THE WHILE LOOP

The C programming language has several structures for looping and conditional branching. We will cover them all in this chapter and we will begin with the while loop. The while loop continues to loop while some condition is true. When the condition becomes false, the looping is discontinued. It therefore does just what it says it does, the name of the loop being very descriptive. Load the program WHILE.C and display it for an example of a while loop. We begin with a comment and the program name, then go on to define an integer variable "count" within the body of the program. The variable is set to zero and we come to the while loop itself.

The syntax of a while loop is just as shown here. The keyword "while" is followed by an expression of something in parentheses, followed by a compound statement bracketed by braces. As long as the expression in parenthesis is true, all statements within the braces will be executed. In this case, since the variable count is incremented by one every time the statements are executed, it will eventually reach 6, the statement will not be executed, and the loop will be terminated. The program control will resume at the statement following the statements in braces. We will cover the compare expression, the one in parentheses, in the next chapter. Until then, simply accept the expressions for what you think they should do and you will probably be correct.

Several things must be pointed out regarding the while loop. First, if the variable count were initially set to any number greater than 5, the statements within the loop would not be executed at all, so it is possible to have a while loop that never is executed. Secondly, if the variable were not incremented in the loop, then in this case, the loop would never

terminate, and the program would never complete. Finally, if there is only one statement to be executed within the loop, it does not need braces but can stand alone. Compile and run this program.

THE DO-WHILE LOOP

A variation of the while loop is illustrated in the program DOWHILE.C, which you should load and display. This program is nearly identical to the last one except that the loop begins with the reserved word "do", followed by a compound statement in braces, then the reserved word "while", and finally an expression in parentheses. The statements in the braces are executed repeatedly as long as the expression in parentheses is true. When the expression in parentheses becomes false, execution is terminated, and control passes to the statements following this statement.

Several things must be pointed out regarding this statement. Since the test is done at the end of the loop, the statements in the braces will always be executed at least once. Secondly, if "i" were not changed within the loop, the loop would never terminate, and hence the program would never terminate. Finally, just like for the while loop, if only one statement will be executed within the loop, no braces are required. Compile and run this program to see if it does what you think it should do. It should come as no surprise to you that these loops can be nested. That is, one loop can be included within the compound statement of another loop, and the nesting level has no limit.

THE FOR LOOP

The "for" loop is really nothing new, it is simply a new way to describe the "while" loop. Load and edit the file named FORLOOP.C for an example of a program with a "for" loop. The "for" loop consists of the reserved word "for" followed by a rather large expression in parentheses. This expression is really composed of three fields separated by semi-colons. The first field contains the expression "index = 0" and is an initializing field. Any expressions in this field are executed prior to the first pass through the loop. There is essentially no limit as to what can go here, but good programming practice would require it to be kept simple. Several initializing statements can be placed in this field, separated by commas.

The second field, in this case containing "index < 6", is the test which is done at the beginning of each loop through the program. It can be any expression which will evaluate to a true or false. (More will be said about the actual value of true and false in the next chapter.) The expression contained in the third field is executed each time the loop is executed but it is not executed until after those statements in the main body of the loop are executed. This field, like the first, can also be composed of several operations separated by commas.

Following the for() expression is any single or compound statement which will be executed as the body of the loop. A compound statement is any group of valid C statements enclosed in braces. In nearly any context in C, a simple statement can be replaced by a compound statement that will be treated as if it were a single statement as far as program control goes. Compile and run this program.

THE IF STATEMENT

Load and display the file IFELSE.C for an example of our first conditional branching statement, the "if". Notice first, that there is a "for" loop with a compound statement as its executable part containing two "if" statements. This is an example of how statements can be nested. It should be clear to you that each of the "if" statements will be executed 10 times. Consider the first "if" statement. It starts with the keyword "if" followed by an expression in parentheses. If the expression is evaluated and found to be true, the single statement following the "if" is executed, and if false, the following statement is skipped. Here too, the single statement can be replaced by a compound statement composed of several statements bounded by braces.

The expression "data == 2" is simply asking if the value of data is equal to 2, this will be explained in detail in the next chapter. (Simply suffice for now that if "data = 2" were used in this context, it would mean a completely different thing.) NOW FOR THE IF-ELSE The second "if" is similar to the first with the addition of a new reserved word, the "else" following the first printf statement. This simply says that if the expression in the parentheses evaluates as true, the first expression is executed, otherwise the expression following the "else" is executed. Thus, one of the two expressions will always be executed, whereas in the first example the single expression was either executed or skipped. Both will find many uses in your C

programming efforts. Compile and run this program to see if it does what you expect.

THE BREAK AND CONTINUE

Load the file named BREAKCON.C for an example of two new statements. Notice that in the first "for", there is an if statement that calls a break if xx equals 9. The break will jump out of the loop you are in and begin executing statements following the loop, effectively terminating the loop. This is a valuable statement when you need to jump out of a loop depending on the value of some results calculated in the loop. In this case, when xx reaches 8, the loop is terminated and the last value printed will be the previous value, namely 7. The next "for" loop, contains a continue statement which does not cause termination of the loop but jumps out of the present iteration. When the value of xx reaches 8 in this case, the program will jump to the end of the loop and continue executing the loop, effectively eliminating the printf statement during the pass through the loop when xx is eight. Compile and run the program to see if it does what you expect.

THE SWITCH STATEMENT

Load and display the file SWITCH.C for an example of the biggest construct yet in the C language, the switch. The switch is not difficult, so don't let it intimidate you. It begins with the keyword "switch" followed by a variable in parentheses which is the switching variable, in this case "truck". As many cases as desired are then enclosed within a pair of braces. The reserved word "case" is used to begin each case entered followed by the value of the variable, then a colon, and the statements to be executed. In this example, if the variable "truck" contains the value 3 during this pass of the switch statement, the printf will cause "The value is three" to be displayed, and the "break" statement will cause us to jump out of the switch.

Once an entry point is found, statements will be executed until a "break" is found or until the program drops through the bottom of the switch braces. If the variable has the value 5, the statements will begin executing where "case 5 :" is found, but the first statements found are where the case 8 statements are. These are executed and the break statement in the "case 8" portion will direct the execution out the bottom of the switch. The various case values can be in any order and if a value is not found, the default portion of the switch will be executed. It should be clear that any of the above constructs can be nested within each other or placed in succession, depending on the needs of the particular programming project at hand.

Compile and run SWITCH.C to see if it does what you expect it to after this discussion. Load and display the file GOTOEX.C for an example of a file with some "goto" statements in it. To use a "goto" statement, you simply use the reserved word "goto" followed by the symbolic name to which you wish to jump. The name is then placed anywhere in the program followed by a colon. You are not allowed to jump into any loop, but you are allowed to jump out of a loop. Also, you are not allowed to jump out of any function into another. These attempts will be flagged by your compiler as an error if you attempt any of them. This particular program is really a mess but it is a good example of why software writers are trying to eliminate the use of the "goto" statement as much as possible.

The only place in this program where it is reasonable to use the "goto" is the one in line 17 where the program jumps out of the three nested loops in one jump. In this case it would be rather messy to set up a variable and jump successively out of all three loops but one "goto" statement gets you out of all three. Some persons say the "goto" statement should never be used under any circumstances but this is rather narrow minded thinking. If there is a place where a "goto" will clearly do a neater control flow than some other construct, feel free to use it. It should not be abused however, as it is in the rest of the program on your monitor. Entire books are written on "gotoless" programming, better known as Structured Programming. These will be left to your study.

One point of reference is the Visual Calculater described in Chapter 14 of this tutorial. This program is contained in four separately compiled programs and is a rather large complex program. If you spend some time studying the source code, you will find that there is not a single "goto" statement anywhere in it. Compile and run GOTOEX.C and study its output. It would be a good exercise to rewrite it and see how much more readable it is when the statements are listed in order.

FINALLY, A MEANINGFUL PROGRAM

Load the file named TEMPCONV.C for an example of a useful, even though somewhat limited program. This is a program that generates a list of centigrade and farenheit temperatures and prints a message out at the freezing point of water and another at the boiling point of water. Of particular importance is the formatting. The header is simply several lines of comments describing what the program does in a manner that catches the readers attention and is still pleasing to the eye.

You will eventually develop your own formatting style, but this is a good way to start. Also if you observe the for loop, you will notice that all of the contents of the compound statement are indented 3 spaces to the right of the "for" reserved word, and the closing brace is lined up under the "f" in "for". This makes debugging a bit easier because the construction becomes very obvious. You will also notice that the "printf" statements that are in the "if" statements within the big "for" loop are indented three additional spaces because they are part of another construct. This is the first program in which we used more than one variable. The three variables are simply defined on three different lines and are used in the same manner as a single variable was used in previous programs. By defining them on different lines, we have an opportunity to define each with a comment.

ANOTHER POOR PROGRAMMING EXAMPLE

Recalling UGLYFORM.C from the last chapter, you saw a very poorly formatted program. If you load and display DUMBCONV.C you will have an example of poor formatting which is much closer to what you will actually find in practice. This is the same program as TEMPCONV.C with the comments removed and the variable names changed to remove the descriptive aspect of the names. Although this program does exactly the same as the last one, it is much more difficult to read and understand. You should begin to develop good programming practices now. Compile and run this program to see that it does exactly what the last one did.

PROGRAMMING EXERCISES

1. Write a program that writes your name on the monitor ten times. Write this program three times, once with each looping method.
2. Write a program that counts from one to ten, prints the values on a separate line for each, and includes a message of your choice when the count is 3 and a different message when the count is 7.

oooooooooo0000000000ooooooooo

An index of Rainbow OS9 articles
compiled by Bob Devries

January 1984 page 132
KISSable OS9 - Nostalgia and notes
Dale L. Puckett

February 1984 page 332
The Advanced Operator - Free up more OS9 workspace
Frank Hogg

February 1984 page 324
KISSable OS9 - Some technical potpourri
Dale L. Puckett

March 1984 page 293
hogg_wash - A new language from England
Frank Hogg

March 1984 page 298
KISSable OS9 - Some technical potpourri
Dale L. Puckett

March 1984 page 291
One Disk - Combining your OS9 boot and system disks
Melvin Hefter

April 1984 page 306
KISSable OS9 - Some technical potpourri
Dale L. Puckett

May 1984 page 314
hogg_wash - Comparing FLEX and OS9
Frank Hogg

May 1984 page 297
KISSable OS9 - Some technical potpourri
Dale L. Pucket

June 1984 page 278
hogg_wash - A FLEXible choice
Frank Hogg

AUSTRALIAN OS9 USER GROUP APPLICATION FORM

****

Surname : _____ First Name :_____ Title (Mr.,Dr.,etc) : ____

Street : _____

City / Suburb : _____ State :___ Postcode : ____

Home Phone :_____ Business Phone : _____

Age Group (please tick) Under 18 [__] 18-25 [__] 26-35 [__] 36-45 [__] 46-55 [__] over 55 [__]

Do you run OS9 Level 1 [_] OS9 Level 2 [_] OSK [_] (please tick)

Type of Computer for OS9 :_____Memory Size :_____

Diskette Size (inches): __ Number of Cylinders (tracks per side): __ Sides:__ Number of Drives :__

Printer Type: _____

Modem Type : _____

Other hardware : _____

Special Interests : _____

Can you contribute articles to this Newsletter :_____

Date : __/__/__ Signature :_____

Amount Enclosed: $ _____ (**$18-00** will cover you for **12 months**)
(A$25-00 for overseas subscriptions)

Cheques made payable to NATIONAL OS9 USERGROUP please.

All subscriptions start with the September issue, and back issues will be sent where applicable.

Please Return Completed Form to :-

NATIONAL OS9 USER GROUP
C/o Gordon BENTZEN
8 ODIN STREET,
SUNNYBANK QLD 4109 (Phone 07 344 3881)
AUSTRALIA